1. The NIST Special Publication 800-63B

If you – 50 years ago – needed to come up with a secret password you were probably part of a secret espionage organization or (more likely) you were pretending to be a spy when playing as a kid. Today, many of us are forced to come up with new passwords *all the time* when signing into sites and apps. As a password *inventeur* it is your responsibility to come up with good, hard-to-crack passwords. But it is also in the interest of sites and apps to make sure that you use good passwords. The problem is that it's really hard to define what makes a good password. However, *the National Institute of Standards and Technology* (NIST) knows what the second best thing is: To make sure you're at least not using a *bad* password.

In this notebook, we will go through the rules in NIST Special Publication 800-63B (https://pages.nist.gov/800-63-3/sp800-63b.html) which details what checks a *verifier* (what the NIST calls a second party responsible for storing and verifying passwords) should perform to make sure users don't pick bad passwords. We will go through the passwords of users from a fictional company and use R to flag the users with bad passwords. But us being able to do this already means the fictional company is breaking one of the rules of 800-63B:

> Verifiers SHALL store memorized secrets in a form that is resistant to offline attacks. Memorized secrets SHALL be salted and hashed using a suitable one-way key derivation function.

That is, never save users' passwords in plaintext, always encrypt the passwords! Keeping this in mind for the next time we're building a password management system, let's load in the data.

*Warning: The list of passwords and the fictional user database both contain **real** passwords leaked from **real** websites. These passwords have not been filtered in any way and include words that are explicit, derogatory and offensive.*

In [89]:
```
# Importing the tidyverse library
library(tidyverse)


# Loading in datasets/users.csv
users <- read_csv('datasets/users.csv')

# Counting how many users we've got
length(users$user_name)


# Taking a look at the 12 first users
 users$user_name[1:12]
```

```
Parsed with column specification:
cols(
  id = col_integer(),
  user_name = col_character(),
  password = col_character()
)
```

982

'vance.jennings' 'consuelo.eaton' 'mitchel.perkins' 'odessa.vaughan' 'araceli.wilder' 'shawn.harrington' 'evelyn.gay' 'noreen.hale' 'gladys.ward' 'brant.zimmerman' 'leanna.abbott' 'milford.hubbard'

In [90]:
```
library(testthat)
library(IRkernel.testthat)
run_tests({
    test_that("Read in data correctly.", {
        expect_is(users, "tbl_df",
            info = 'You should use read_csv (with an underscore) to read "data
sets/users.csv" into users')
    })

    test_that("Read in data correctly.", {
        correct_users <- read_csv('datasets/users.csv')
        expect_equivalent(users, correct_users,
            info = 'users should contain the data in "datasets/users.csv"')
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.125 0.167 2785.328 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

2. Passwords should not be too short

If we take a look at the first 12 users above we already see some bad passwords. But let's not get ahead of ourselves and start flagging passwords *manually*. What is the first thing we should check according to the NIST Special Publication 800-63B?

> Verifiers SHALL require subscriber-chosen memorized secrets to be at least 8 characters in length.

Ok, so the passwords of our users shouldn't be too short. Let's start by checking that!

In [91]:
```
# Calculating the lengths of users' passwords
users$length <- nchar(users$password)

# Flagging the users with too short passwords
users$too_short <- str_length(users$password)<8

# Counting the number of users with too short passwords
sum(users$too_short==TRUE)

# Taking a look at the 12 first rows
users[1:12,]
```

376

id	user_name	password	length	too_short
1	vance.jennings	joobheco	8	FALSE
2	consuelo.eaton	0869347314	10	FALSE
3	mitchel.perkins	fabypotter	10	FALSE
4	odessa.vaughan	aharney88	9	FALSE
5	araceli.wilder	acecdn3000	10	FALSE
6	shawn.harrington	5278049	7	TRUE
7	evelyn.gay	master	6	TRUE
8	noreen.hale	murphy	6	TRUE
9	gladys.ward	lwsves2	7	TRUE
10	brant.zimmerman	1190KAREN5572497	16	FALSE
11	leanna.abbott	aivlys24	8	FALSE
12	milford.hubbard	hubbard	7	TRUE

In [92]:
```
run_tests({
    test_that("The correct number of users are flagged", {
    sum(str_length(users$password) < 8)

    expect_equal(sum(str_length(users$password) < 8), sum(users$too_short),
        info = "users$too_short should be a TRUE/FALSE column where all rows w
ith passwords < 8 are TRUE.")
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.169 0.167 2785.371 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

3. Common passwords people use

Already this simple rule flagged a couple of offenders among the first 12 users. Next up in Special Publication 800-63B is the rule that

> verifiers SHALL compare the prospective secrets against a list that contains values known to be commonly-used, expected, or compromised.
>
> - Passwords obtained from previous breach corpuses.
> - Dictionary words.
> - Repetitive or sequential characters (e.g. ‘aaaaaa’, ‘1234abcd’).
> - Context-specific words, such as the name of the service, the username, and derivatives thereof.

We're going to check these in order and start with *Passwords obtained from previous breach corpuses*, that is, websites where hackers have leaked all the users' passwords. As many websites don't follow the NIST guidelines and encrypt passwords there now exist large lists of the most popular passwords. Let's start by loading in the 10,000 most common passwords which I've taken from here (https://github.com/danielmiessler/SecLists/tree/master/Passwords).

In [93]:
```
# Reading in the top 10000 passwords
common_passwords <- read_lines("datasets/10_million_password_list_top_10000.txt")

# Taking a look at the top 100
#common_passwords[1:100]
```

In [94]:
```
run_tests({
    correct_common_passwords <- read_lines("datasets/10_million_password_list_
top_10000.txt")
    test_that("the data read in is correct", {
    expect_equal(correct_common_passwords, common_passwords,
        info = "datasets/10_million_password_list_top_10000.txt should be read
 in using read_lines and put into common_passwords.")
    })

})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.201 0.171 2785.407 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

4. Passwords should not be common passwords

The list of passwords was ordered, with the most common passwords first, and so we shouldn't be surprised to see passwords like `123456` and `qwerty` above. As hackers also have access to this list of common passwords, it's important that none of our users use these passwords!

Let's flag all the passwords in our user database that are among the top 10,000 used passwords.

In [95]:
```
# Flagging the users with passwords that are common passwords
users$common_password = users$password %in% common_passwords

# Counting the number of users using common passwords
sum(users$common_password==TRUE)

# Taking a look at the 12 first rows
users[1:12,]
```

129

id	user_name	password	length	too_short	common_password
1	vance.jennings	joobheco	8	FALSE	FALSE
2	consuelo.eaton	0869347314	10	FALSE	FALSE
3	mitchel.perkins	fabypotter	10	FALSE	FALSE
4	odessa.vaughan	aharney88	9	FALSE	FALSE
5	araceli.wilder	acecdn3000	10	FALSE	FALSE
6	shawn.harrington	5278049	7	TRUE	FALSE
7	evelyn.gay	master	6	TRUE	TRUE
8	noreen.hale	murphy	6	TRUE	TRUE
9	gladys.ward	lwsves2	7	TRUE	FALSE
10	brant.zimmerman	1190KAREN5572497	16	FALSE	FALSE
11	leanna.abbott	aivlys24	8	FALSE	FALSE
12	milford.hubbard	hubbard	7	TRUE	FALSE

In [96]:
```
run_tests({
    test_that("the number of flagged passwords is correct", {
    expect_equal(sum(users$password %in% common_passwords), sum(users$common_p
assword),
        info = "users$common_password should be TRUE for each row with a passw
ord that is also in common_passwords.")
    })
})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.24 0.171 2785.445 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

5. Passwords should not be common words

Ay ay ay! It turns out many of our users use common passwords, and of the first 12 users there are already two. However, as most common passwords also tend to be short, they were already flagged as being too short. What is the next thing we should check?

> Verifiers SHALL compare the prospective secrets against a list that contains [...] dictionary words.

This follows the same logic as before: It is easy for hackers to check users' passwords against common English words and therefore common English words make bad passwords. Let's check our users' passwords against the top 10,000 English words from Google's Trillion Word Corpus (https://github.com/first20hours/google-10000-english).

In [97]:
```
words <- read_lines("datasets/google-10000-english.txt")

# Flagging the users with passwords that are common words
users$common_word <- users$password %in% words

# Counting the number of users using common words as passwords
sum(users$common_password==TRUE)

# Taking a look at the 12 first rows
users[1:12,]
```

129

id	user_name	password	length	too_short	common_password	commo
1	vance.jennings	joobheco	8	FALSE	FALSE	FALSE
2	consuelo.eaton	0869347314	10	FALSE	FALSE	FALSE
3	mitchel.perkins	fabypotter	10	FALSE	FALSE	FALSE
4	odessa.vaughan	aharney88	9	FALSE	FALSE	FALSE
5	araceli.wilder	acecdn3000	10	FALSE	FALSE	FALSE
6	shawn.harrington	5278049	7	TRUE	FALSE	FALSE
7	evelyn.gay	master	6	TRUE	TRUE	TRUE
8	noreen.hale	murphy	6	TRUE	TRUE	TRUE
9	gladys.ward	lwsves2	7	TRUE	FALSE	FALSE
10	brant.zimmerman	1190KAREN5572497	16	FALSE	FALSE	FALSE
11	leanna.abbott	aivlys24	8	FALSE	FALSE	FALSE
12	milford.hubbard	hubbard	7	TRUE	FALSE	FALSE

In [98]:
```
run_tests({
    correct_words <- read_lines("datasets/google-10000-english.txt")
    test_that("google-10000-english.txt is read in correctly", {
        expect_equal(correct_words, words,
            info = "datasets/google-10000-english.txt should be read in using
 read_lines and put into words.")
    })

    test_that("the number of flagged passwords is correct", {
        users$common_word <- str_to_lower(users$password) %in% words
        expect_equal(sum(users$common_word), sum(str_to_lower(users$password)
%in% correct_words),
            info = "users$common_word should be TRUE for each row with a passw
ord that is also in words.")
    })

})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.282 0.179 2785.494 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

6. Passwords should not be your name

It turns out many of our passwords were common English words too! Next up on the NIST list:

> Verifiers SHALL compare the prospective secrets against a list that contains [...] context-specific words, such as the name of the service, the username, and derivatives thereof.

Ok, so there are many things we could check here. One thing to notice is that our users' usernames consist of their first names and last names separated by a dot. For now, let's just flag passwords that are the same as either a user's first or last name.

In [99]:
```
# Extracting first and last names into their own columns
users$first_name <- str_extract(users$user_name,"^\\w+")
users$last_name <- str_extract(users$user_name,'\\w+$')

# Flagging the users with passwords that matches their names
users$uses_name <- (users$password == users$first_name | users$password == use
rs$last_name)

# Counting the number of users using names as passwords
sum(users$uses_name==TRUE)

# Taking a look at the 12 first rows
users[1:12,]
```

50

id	user_name	password	length	too_short	common_password	commo
1	vance.jennings	joobheco	8	FALSE	FALSE	FALSE
2	consuelo.eaton	0869347314	10	FALSE	FALSE	FALSE
3	mitchel.perkins	fabypotter	10	FALSE	FALSE	FALSE
4	odessa.vaughan	aharney88	9	FALSE	FALSE	FALSE
5	araceli.wilder	acecdn3000	10	FALSE	FALSE	FALSE
6	shawn.harrington	5278049	7	TRUE	FALSE	FALSE
7	evelyn.gay	master	6	TRUE	TRUE	TRUE
8	noreen.hale	murphy	6	TRUE	TRUE	TRUE
9	gladys.ward	lwsves2	7	TRUE	FALSE	FALSE
10	brant.zimmerman	1190KAREN5572497	16	FALSE	FALSE	FALSE
11	leanna.abbott	aivlys24	8	FALSE	FALSE	FALSE
12	milford.hubbard	hubbard	7	TRUE	FALSE	FALSE

In [100]:
```
run_tests({
    correct_first_name <- str_extract(users$user_name, "^\\w+")
    correct_last_name <- str_extract(users$user_name, "\\w+$")

    # Flagging the users with passwords that matches their names
    correct_uses_name <- str_to_lower(users$password) == correct_first_name |
                         str_to_lower(users$password) == correct_last_name
    test_that("the number of flagged passwords is correct", {
        expect_equal(sum(correct_uses_name), sum( users$uses_name),
            info = "users$uses_name should be TRUE for each row with a passwor
d which is also the first or last name.")
    })

})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.336 0.179 2785.547 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

7. Passwords should not be repetitive

Milford Hubbard (user number 12 above), what where you thinking!? Ok, so the last thing we are going to check is a bit tricky:

> verifiers SHALL compare the prospective secrets [so that they don't contain] repetitive or sequential characters (e.g. ‘aaaaaa’, ‘1234abcd’).

This is tricky to check because what is *repetitive* is hard to define. Is `11111` repetitive? Yes! Is `12345` repetitive? Well, kind of. Is `13579` repetitive? Maybe not..? To check for *repetitiveness* can be arbitrarily complex, but here we're only going to do something simple. We're going to flag all passwords that contain 4 or more repeated characters.

In [101]:
```
# Splitting the passwords into vectors of single characters
split_passwords <- strsplit(users$password,"")

# Picking out the max number of repeat characters for each password
users$max_repeats <- sapply(split_passwords, function(split_password) {
  max(rle(split_password)$length)
})

# Flagging the passwords with >= 4 repeats
users$too_many_repeats <- lapply(users$max_repeats,max) >=4

# Taking a look at the users with too many repeats
#get index method one ->>> match(users$too_many_repeats,TRUE)
#method 2 >>>> too_many_repeats_index = which(users$too_many_repeats %in% TRU
E)

users[lapply(users$max_repeats,max) >=4,]
```

id	user_name	password	length	too_short	common_password	common_word
147	patti.dixon	555555	6	TRUE	TRUE	FALSE
573	cornelia.bradley	555555	6	TRUE	TRUE	FALSE
645	essie.lopez	11111	5	TRUE	TRUE	FALSE
799	charley.key	888888	6	TRUE	TRUE	FALSE
808	thurman.osborne	rinnnng0	8	FALSE	FALSE	FALSE
942	mitch.ferguson	aaaaaa	6	TRUE	TRUE	FALSE

In [102]:
```
run_tests({
    correct_max_repeats <- sapply(users$password, function(password) {
        split_password <- str_split(password, "")[[1]]
        rle_password <- rle(split_password)
        max(rle_password$lengths)
    })

    test_that("the number of flagged passwords is correct", {
        expect_equal(sum(users$too_many_repeats), sum( users$max_repeats >= 4
),
            info = "users$too_many_repeats should be TRUE for each row with a
 password with 4 or more repeats.")
    })

})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.44 0.179 2785.65 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

8. All together now!

Now we have implemented all the basic tests for bad passwords suggested by NIST Special Publication 800-63B! What's left is just to flag all bad passwords and maybe to send these users an e-mail that strongly suggests they change their password.

In [103]:
```
# Flagging all passwords that are bad
users$bad_password <- users$too_short | users$common_password | users$common_w
ord | users$too_many_repeats | users$uses_name

# Counting the number of bad passwords
sum(users$bad_password==TRUE)


# Looking at the first 100 bad passwords
users[which(users$bad_password %in% TRUE),]
```

424

id	user_name	password	length	too_short	common_password	common_wor
6	shawn.harrington	5278049	7	TRUE	FALSE	FALSE
7	evelyn.gay	master	6	TRUE	TRUE	TRUE
8	noreen.hale	murphy	6	TRUE	TRUE	TRUE
9	gladys.ward	lwsves2	7	TRUE	FALSE	FALSE
12	milford.hubbard	hubbard	7	TRUE	FALSE	FALSE
14	jamie.cochran	310356	6	TRUE	FALSE	FALSE
16	lorrie.gay	oZ4k0QE	7	TRUE	FALSE	FALSE
17	domingo.dyer	chelsea	7	TRUE	TRUE	TRUE
18	martin.pacheco	zvc1939	7	TRUE	FALSE	FALSE
19	shelby.massey	nickgd	6	TRUE	FALSE	FALSE
22	leticia.sanford	cocacola	8	FALSE	TRUE	FALSE
23	jenny.woodard	woodard	7	TRUE	FALSE	FALSE
26	dianna.munoz	AJ9Da	5	TRUE	FALSE	FALSE
27	julia.savage	ewokzs	6	TRUE	FALSE	FALSE
29	joaquin.walters	YyGjz8E	7	TRUE	FALSE	FALSE
31	rosanna.reid	reid	4	TRUE	FALSE	TRUE
35	roger.golden	jOYZBs8	7	TRUE	FALSE	FALSE
39	gus.padilla	wwewwf1	7	TRUE	FALSE	FALSE
44	vito.nicholson	225377	6	TRUE	FALSE	FALSE
46	morris.price	NdZ7E6	6	TRUE	FALSE	FALSE
48	jarred.white	CQB3Z	5	TRUE	FALSE	FALSE
49	junior.burch	diffo	5	TRUE	FALSE	FALSE
52	ahmad.hopper	123456789	9	FALSE	TRUE	FALSE
53	rosario.merrill	y8uM7D6	7	TRUE	FALSE	FALSE
57	dwayne.hurst	mikeloo	7	TRUE	FALSE	FALSE
59	rosalinda.rodriquez	golden	6	TRUE	TRUE	TRUE
60	landon.middleton	apr2502	7	TRUE	FALSE	FALSE
63	dominique.dawson	jfasii	6	TRUE	FALSE	FALSE
64	reuben.higgins	sanaang	7	TRUE	FALSE	FALSE
65	ursula.wood	jecarn	6	TRUE	FALSE	FALSE
...	...	...	...	...	...	...
915	bryant.briggs	briggs	6	TRUE	TRUE	FALSE

id	user_name	password	length	too_short	common_password	common_wor
918	amparo.evans	pedir5	6	TRUE	FALSE	FALSE
919	trevor.perkins	iX69N	5	TRUE	FALSE	FALSE
920	julie.lloyd	dwihen	6	TRUE	FALSE	FALSE
921	alexandria.hinton	hhdmrd.	7	TRUE	FALSE	FALSE
923	arthur.buck	GoSWy	5	TRUE	FALSE	FALSE
925	jerrold.rodriquez	eBSy8	5	TRUE	FALSE	FALSE
926	houston.garcia	PwXM2K	6	TRUE	FALSE	FALSE
928	roberto.torres	2277272	7	TRUE	FALSE	FALSE
931	davis.roth	angel	5	TRUE	TRUE	TRUE
932	abdul.rowland	8685041	7	TRUE	FALSE	FALSE
933	marlin.fleming	bgeesh1	7	TRUE	FALSE	FALSE
935	evan.chan	hunter	6	TRUE	TRUE	TRUE
940	lee.hendricks	lee	3	TRUE	FALSE	TRUE
941	zane.bond	pepper	6	TRUE	TRUE	TRUE
942	mitch.ferguson	aaaaaa	6	TRUE	TRUE	FALSE
943	donna.mcmahon	chicago	7	TRUE	TRUE	TRUE
944	andrea.day	bigdog	6	TRUE	TRUE	FALSE
949	annie.mendez	manc13	6	TRUE	FALSE	FALSE
951	rebekah.sharpe	money	5	TRUE	TRUE	TRUE
953	christa.morrison	mercedes	8	FALSE	TRUE	TRUE
957	devon.holman	raiders	7	TRUE	TRUE	FALSE
961	carmelo.byers	asdfgh	6	TRUE	TRUE	FALSE
965	chrystal.burns	DILWYN	6	TRUE	FALSE	FALSE
966	irma.vasquez	spider	6	TRUE	TRUE	TRUE
967	taylor.kent	summer	6	TRUE	TRUE	TRUE
968	deloris.dixon	seeks	5	TRUE	FALSE	TRUE
971	noel.montoya	colours	7	TRUE	FALSE	TRUE
976	freeman.rose	rangers	7	TRUE	TRUE	TRUE
981	lora.quinn	antonau	7	TRUE	FALSE	FALSE

In [104]:
```
run_tests({
    correct_bad_password <- users$too_short | users$common_word |
                        users$common_password | users$uses_name |
                        users$too_many_repeats
    test_that("all the bad passwords are flagged", {
    expect_equal(sum(correct_bad_password), sum(users$bad_password),
        info = "All rows with passwords that should be flagged as bad should h
ave users$bad_password set to TRUE.")
    })

})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.558 0.179 2785.768 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```

9. Otherwise, the password should be up to the user

In this notebook, we've implemented the password checks recommended by the NIST Special Publication 800-63B. It's certainly possible to better implement these checks, for example, by using a longer list of common passwords. Also note that the NIST checks in no way guarantee that a chosen password is good, just that it's not obviously bad.

Apart from the checks we've implemented above the NIST is also clear with what password rules should *not* be imposed:

> Verifiers SHOULD NOT impose other composition rules (e.g., requiring mixtures of different character types or prohibiting consecutively repeated characters) for memorized secrets. Verifiers SHOULD NOT require memorized secrets to be changed arbitrarily (e.g., periodically).

So the next time a website or app tells you to "include both a number, symbol and an upper and lower case character in your password" you should send them a copy of NIST Special Publication 800-63B (https://pages.nist.gov/800-63-3/sp800-63b.html).

In [105]:
```
# Enter a password that passes the NIST requirements
# PLEASE DO NOT USE AN EXISTING PASSWORD HERE
new_password <- "samppy2361"
```

In [106]:
```
run_tests({
    temp_common_passwords <- read_lines("datasets/10_million_password_list_top
_10000.txt")
    temp_words <- read_lines("datasets/google-10000-english.txt")

    is_bad <- str_length(new_password) < 8 |
        new_password %in% temp_common_passwords |
        str_to_lower(new_password) %in% temp_words |
        max(rle(str_split(new_password, "")[[1]])$lengths) >= 4

    test_that("", {
    expect_false(is_bad,
        info = "This password does not fulfill the NIST requirements.")
    })

})
```

```
<ProjectReporter>
  Inherits from: <ListReporter>
  Public:
    .context: NULL
    .end_context: function (context)
    .start_context: function (context)
    add_result: function (context, test, result)
    all_tests: environment
    cat_line: function (...)
    cat_tight: function (...)
    clone: function (deep = FALSE)
    current_expectations: environment
    current_file: some name
    current_start_time: 5.597 0.179 2785.806 0.005 0
    dump_test: function (test)
    end_context: function (context)
    end_reporter: function ()
    end_test: function (context, test)
    get_results: function ()
    initialize: function (...)
    is_full: function ()
    out: 3
    results: environment
    rule: function (...)
    start_context: function (context)
    start_file: function (name)
    start_reporter: function ()
    start_test: function (context, test)
```